Revista Brasileira de Ensino de Física
https://doi.org/10.1590/1806-9126-RBEF-2025-0133

Material Suplementar para "Algoritmo de Refração: Uma proposta didática para investigação do problema de minimização do tempo"

Anexo B - Algoritmo de Refração: O Problema De Minimização Do Tempo

O Princípio de Fermat foi implementado utilizando a linguagem *Octave*, um software livre desenvolvido para computação matemática e licenciado sob a GNU *General Public License* (GPL). Sua licença livre torna o *Octave* uma ferramenta acessível e versátil, adequada para diferentes públicos e contextos educacionais, promovendo a democratização do acesso a recursos computacionais avançados.

O algoritmo desenvolvido permite calcular os comprimentos de diversas trajetórias, obter os tempos de percurso do raio luminoso para cada um destes possíveis caminhos, ordená-los crescentemente em termos destes tempos e verificar a consistência dos resultados com a Lei de Snell-Descartes. Desta forma, conseguimos explorar a minimização do tempo de percurso da luz em meios com diferentes índices de refração, tal como proposto por Fermat.

A implementação do algoritmo foi organizada de forma modular, incluindo o cálculo de distâncias, tempos e trajetórias, além de rotinas para gerar visualizações gráficas, facilitando a exploração do código e permitindo a realização de testes, ajustes e adaptação da ferramenta a diferentes contextos de aplicação. As etapas podem ser resumidas conforme abaixo.

Inicialização – identificação das variáveis e constantes:

- Registro das coordenadas fixas dos pontos P e Q;
- Valores dos índices de refração dos meios 1 e 2;
- Registro do valor da velocidade da luz no vácuo;
- Parâmetros de discretização do problema;
- Definição das listas/vetores para gravação dos valores gerados.

Iteração:

- A partir dos valores indicados de x_i, calcular e registrar:
  - Distâncias percorridas em cada trajetória;
  - Tempos de percurso.

Identificar/otimização:

- Ordenar a lista de caminhos segundo o menor tempo e identificar o valor de x_i.

Gráficos e comparação com a Lei de Snell-Descartes:

- Preparar a área da primeira figura, relativa a cada um dos caminhos calculados;
- Gerar linhas relativas a cada um dos caminhos;
- Gerar a primeira figura ilustrativa com os caminhos calculados e salvá-la;
- Gerar o gráfico de tempo de percurso versus x_i e salvá-la;
- Calcular os ângulos θ_1 e θ_2 a partir do valor mínimo de x_i;
- Calcular $sen\ (\theta_1)/sen\ (\theta_2)$ e comparar com a relação n_2/n_1.

Nas tabelas a seguir são apresentadas o código utilizado na implementação do algoritmo, com os detalhamentos das linhas e suas funções no código. É importante destacar que textos precedidos pelo símbolo '%' são comentários, ou seja, não são processados pelo programa; eles têm como objetivo organizar e esclarecer o funcionamento do código. Além disto, o caractere ';' no final de cada linha indica que o cálculo não será mostrado na tela.

No Quadro B1 são apresentadas as linhas iniciais do código. A linha 1 contém comandos para limpar o ambiente de trabalho: `clear` remove todas as variáveis da memória e `clc` limpa o *console*. Nas linhas 2 e 3 são definidos os pontos P e Q, utilizados no problema, com coordenadas cartesianas $P = (-1{,}1)\ m$ e $Q = (1, -1)\ m$. As linhas 4 e 5 estabelecem os índices de refração n_1 e n_2 dos meios envolvidos, assumindo valores correspondentes ao ar e a água (1,00 e 1,333, respectivamente). Na linha 6 é atribuído o valor da velocidade da luz no vácuo: $c = 299.792.458\ m/s$. Esses parâmetros de entrada são essenciais para a simulação e resolução do problema de minimização do tempo de percurso da luz entre dois pontos.

Quadro B1 - Implementação inicial do código.

Linha	Implementação	Linha	Implementação
1	`clear, clc;`	**7**	`numero = 20;`
2	`p = [-1, 1];`	**8**	`passo = 1/numero;`
3	`q = [1, -1];`	**9**	`xi = [];`
4	`n1 = 1.00;`	**10**	`tempo = [];`
5	`n2 = 1.333;`	**11**	`lista = [];`
6	`c = 299792458 % m/s`		

Ainda no Quadro B1, do lado direito, a linha 7 define o número de valores de x_i a serem avaliados, caracterizando a discretização do problema. Neste exemplo, foi adotado o valor 20. Por convenção, o intervalo de estudo para x_i está limitado pelas abscissas dos pontos P e Q, ou seja, entre –1 e 1. Na linha 8, estabelece-se o passo da variação de x_i, de modo que o código avalia x_i em $2 \cdot numero + 1$ pontos, resultando em uma resolução de 0,05 metros por passo.

As linhas 9 e 10 inicializam listas vazias para armazenar os valores de x_i e os tempos totais de percurso t, que serão usados para gerar o gráfico de tempo versus posição x_i. Esse gráfico é essencial para visualizar o valor 'ótimo' de x_i, associado ao tempo mínimo. A linha 11 também inicializa uma lista vazia, destinada a armazenar os valores das distâncias totais de percurso, além de registrar os dados de x_i e t que alimentam o restante do algoritmo.

O lado esquerdo do Quadro B2 apresenta o trecho do código entre as linhas 12 e 19. Na linha 12, é iniciado um laço for, que itera a variável i no intervalo simétrico de `−numero` até `numero`, conforme a sintaxe especificada. Esse loop é responsável por calcular os valores de x_i a cada passo, com `xi = i * passo por definição`. Esses valores são então inseridos ao final da lista `xi`, conforme registrado na linha 13.

A linha 14 define as coordenadas temporárias do ponto R, onde a ordenada é sempre nula e a abscissa é o valor de x_i. Na linha 15, o código instrui o Octave a exibir no console o valor atual de x_i por meio da função `printf()`. Os argumentos dessa função especificam a

saída formatada: o texto `'xi = '` seguido de `%d\n`, que imprime o valor numérico com quebra de linha. Esse comando tem finalidade ilustrativa, servindo como acompanhamento da execução.

As linhas 16 e 17 realizam os cálculos das velocidades da luz nos meios 1 e 2 utilizando a Equação (1), que relaciona índice de refração e velocidade de propagação. Por fim, nas linhas 18 e 19, são calculadas as distâncias S_1 e S_2, utilizando as Equações (5) e (6). Os índices usados nos vetores `p()` e `o()` referem-se às coordenadas cartesianas: 1 indica a abscissa (x) e 2 a ordenada (y).

Quadro B2 - Loop com os cálculos dos tempos de cada uma das trajetórias determinadas.

Linha	Implementação	Linha	Implementação
12	`for i = -numero: numero`	**20**	`S = s1+s2;`
13	`xi = [xi; i*passo];`	**21**	`t1 = s1/v1;`
14	`O = [i*passo, 0];`	**22**	`t2 = s2/v2;`
15	`printf("xi = %d\n", i* passo);`	**23**	`t = t1+t2;`
16	`v1 = c/n1;`	**24**	`tempo = [tempo;t];`
17	`v2 = c/n2;`	**25**	`lista = [lista;i*passo,t,s];`
18	`s1 = sqrt((p(1)-o(1))^2 + (p(2)-o(2))^2);`	**26**	`printf("---------- -- \n");`
19	`s2 = sqrt((q(1)-o(1))^2 + (q(2)-o(2))^2);`	**27**	`end`
		28	`final = size(xi)(1)`

A primeira linha do lado direito do Quadro B2, linha 20, calcula a distância total do percurso luminoso, obtida pela soma das distâncias previamente calculadas: S_1 e S_2. Nas linhas 21 e 22, são determinados os tempos t_1 e t_2 correspondentes a cada trecho, com base nas velocidades v_1 e v_2 da luz nos respectivos meios. O tempo total de percurso entre os pontos P e Q, passando por R, é computado na linha 23 como a soma desses dois intervalos de tempo.

O valor obtido é então armazenado ao final da lista `tempo` (linha 24). Na sequência, o algoritmo registra os valores calculados — `xi`, `t` e `s` — no vetor `lista`, conforme a linha 25.

Para facilitar a visualização dos dados durante a execução do laço, a linha 26 imprime no console um separador indicativo. O laço for é encerrado na linha 27 com o comando `end`.

Por fim, a linha 28 define a variável final com o total de elementos calculados, utilizando o comando `size(xi)(1)`, que retorna o número de linhas do vetor `xi`. Esse número corresponde à quantidade de valores de x_i avaliados na discretização. Por curiosidade, a função `size()` retorna uma matriz com duas dimensões: a primeira representa o número de linhas e a segunda, o número de colunas.

O Quadro B3 apresenta o trecho do código responsável pela construção da área gráfica onde são representados os caminhos utilizados nos cálculos. A linha 29 indica que a figura identificada como `(1)` será editada, enquanto a linha 30 limpa qualquer gráfico anterior, abrindo uma nova janela gráfica. As linhas 31 e 32 desenham linhas pontilhadas horizontais e verticais, que servem para destacar os eixos x e y. Para fins didáticos, a figura foi configurada com os pontos $P = [-1,1]$ e $Q = [1,-1]$.

Quadro B3 - Determinando a área da primeira figura, onde serão apresentados os caminhos dos percorridos pelos feixes da luz.

Linha	Implementação
29	`figure (1);`
30	`clf;`
31	`line([-1.5 1.5], [0 0], "linestyle", "-.", "color", "k");`
32	`line([0 0], [-1.5 1.5], "linestyle", "-.", "color", "k");`
33	`line([ -1.5 1.5], [1.5 1.5], "linestyle", "-", "color", "k");`
34	`line([1.5 1.5], [1.5 -1.5], "linestyle", "-", "color", "k");`
35	`text(1,0.5, "Meio 1", "fontsize" ,12)`
36	`text( -1, -0.5, "Meio 2", "fontsize" ,12)`
37	`text( -1.1 ,1.1, "P", "fontsize ", 20);`
38	`text(1, -1.1, "Q", "fontsize ", 20);`
39	`xlabel("Posição x(m)", "fontsize" ,12);`
40	`ylabel("Posição y(m)", "fontsize" ,12);`
41	`title("Caminho da luz entre os pontos 'P' e 'Q'","fontsize",12)`

A automatização da construção da figura, permitindo que ela se adapte a diferentes configurações de entrada, pode ser proposta como um desafio adicional para estudantes interessados em aprofundar seus conhecimentos em programação com o Octave. Entretanto, recomenda-se, em um primeiro momento, que o aluno domine a estrutura do código aplicada aos pontos fixos definidos, antes de buscar generalizações.

O comando `line` possui a seguinte estrutura lógica: os dois primeiros vetores entre colchetes correspondem aos valores inicial e final das coordenadas x e y, que definem os pontos extremos da linha. Em seguida, são especificados os argumentos `linestyle` e `'-.'`, que indicam o estilo da linha (neste caso, traço-ponto). Por fim, os argumentos de cor são indicados — por exemplo, `k` representa a cor preta, remetendo à letra final da palavra *black* em inglês

A linha 33 insere uma linha superior para delimitar o gráfico, enquanto a linha 34 adiciona uma linha inferior, formando uma moldura tracejada com origem centralizada. As linhas 35 e 36 definem as posições das legendas que identificam os meios 1 e 2. Em sequência, as linhas 37 e 38 estabelecem as coordenadas das legendas referentes aos pontos P e Q. Nas linhas 39 e 40, são adicionados os rótulos dos eixos x e y, respectivamente. Por fim, a linha 41 define o título do gráfico.

O comando `text` possui lógica semelhante ao `line`, mas com particularidades. Nele, é necessário apenas indicar a posição (x, y) onde o texto será inserido na figura — esses valores são os dois primeiros argumentos numéricos, separados por vírgula. Após a posição, incluem-se parâmetros como `fontsize` seguido do valor da fonte (por exemplo, 15), que definem o estilo da exibição textual. Por outro lado, os comandos `xlabel` e `ylabel` não exigem a indicação de posição, pois inserem automaticamente os rótulos nos eixos horizontal e vertical, respectivamente, com posicionamento padronizado pela interface gráfica.

O Quadro B4 apresenta o trecho do código correspondente às linhas 42 a 57.
Na linha 42, o vetor lista é ordenado em ordem crescente com base no segundo elemento de cada linha (representando o tempo *t*), por meio do comando `sortrows(lista, 2)`. Para armazenar essa ordenação, é criada a variável `ordem`, que facilita a identificação do caminho

de menor tempo. Na linha 43, define-se a matriz `cores`, com comprimento igual ao número total de caminhos calculados (`final`), utilizando o `colormap jet` — uma paleta de cores do Octave que varia do azul ao vermelho, passando por tons intermediários como ciano e amarelo. Essa matriz será usada para atribuir cores distintas às trajetórias simuladas no gráfico, favorecendo sua diferenciação visual. As linhas 44 a 47 iniciam um laço `for`, com contador `j`, que percorre todos os caminhos simulados. Esse laço é responsável por desenhar no gráfico cada trajetória da luz com uma cor distinta da paleta gerada, permitindo análise comparativa das diferentes rotas.

As linhas 44 e 45 desenham cada uma das trajetórias associadas ao índice `j`. A lógica e a sintaxe seguem os princípios descritos anteriormente: os dois primeiros elementos do argumento definem os pontos extremos da linha, seguidos pelo parâmetro `linestyle`, onde o símbolo `"-"` indica uma linha contínua, e pela cor atribuída pela paleta criada anteriormente. A variável `cores(j, :)` representa a j-ésima cor da matriz gerada pela paleta escolhida. Destaca-se que o valor `ordem(j,1)` corresponde ao j-ésimo valor de x_i, pois tanto a primeira coluna da matriz ordem quanto a da matriz lista armazenam as abscissas utilizadas na simulação.

Convém ressaltar que a linha 44 representa o trajeto S_1 e a linha 45, o trajeto S_2. Já as linhas 48 e 49 destacam o caminho de menor tempo, traçado em preto e com espessura acentuada. Essa característica é definida pelos dois últimos parâmetros do comando, nos quais o atributo `linewidth` é configurado com o valor 2.

Quadro B4 - Implementação gráfica das trajetórias no primeiro gráfico.

Linha	Implementação
42	`ordem = sortrows (lista, 2);`
43	`cores = jet(final);`
44	`for j = 1: final`
45	`line([p(1) ordem(j ,1)], [p(2) o(2)], "linestyle", "-", "color ", cores (j, :));`
46	`line([ordem(j ,1) q(1)], [o(2) q(2)], "linestyle", "-", "color ", cores`

Linha	Implementação
	`(j, :));`
47	`end`
48	`line([p (1) ordem(1 ,1)], [p(2) o(2)], "linestyle", "-", "color", "k", "linewidth", 2);`
49	`line([ordem(1 ,1) q(1) ], [o(2) q(2)], "linestyle", "-", "color", "k", "linewidth", 2);`
50	`colormap(jet)`
51	`h = colorbar();`
52	`ytick = get (h, "ytick");`
53	`escala = [ordem(1,2), ordem(ceil (final/5),2), ordem(ceil (2*final/5),2) , ordem(ceil(3*final/5),2), ordem(ceil(4*final/5),2) ,ordem(final,2) ];`
54	`set(h, "yticklabel", escala);`
55	`ylim([ -1.5 1.5]);`
56	`text(1.9 ,1.65," tempo (s)", "fontsize" ,12)`
57	`saveas(1,"figure1.png")`

Entre as linhas 50 e 55 estão os comandos relativos à escala de cores. Na linha 50, o comando `colormap(jet)` define a mesma paleta de cores utilizada para representar as trajetórias. Em seguida, na linha 51, é adicionada uma barra de cores ao gráfico por meio do comando `colorbar()`, cuja referência é armazenada na variável *h*. Essa barra indica visualmente a correspondência entre as cores exibidas no gráfico e os valores numéricos dos tempos de percurso (*t*).

O comando na linha 52, `get(h, "ytick")`, obtém os valores atuais das marcas de divisão (`ticks`) no eixo (y) da barra de cores. Essa informação é usada para personalizar os rótulos dessas divisões. Na linha 53, a variável escala é definida para representar os valores desejados nas marcas da barra. Foram escolhidas seis posições: o menor e o maior tempo (primeiro e último valores da matriz ordem, segunda coluna), além de quatro valores intermediários, calculados com base em frações do número total de elementos (variável final). A função `ceil()` é utilizada para arredondar os índices para cima, garantindo que os valores

selecionados estejam dentro dos limites da matriz.

Na linha 54, o comando `set(h, "yticklabel", escala)` substitui os rótulos padrão da barra de cores pelos valores definidos em escala, permitindo realçar os tempos de percurso de forma personalizada. Já a linha 55, por meio da função `ylim([-1.5 1.5])`, ajusta os limites de visualização vertical do gráfico, assegurando que a barra de cores fique enquadrada corretamente. A linha 56 insere uma legenda explicativa usando `text`, e por fim, a linha 57 salva a figura como imagem, com o comando `saveas(1,"figure1.png")`, onde "`1`" refere-se ao número da figura, e `figure1.png` é o nome do arquivo gerado.

No Quadro B5, um segundo gráfico é gerado entre as linhas 58 e 64. Este gráfico representa a relação entre o tempo total de percurso t e a posição x_i. Assim como no gráfico anterior, a linha 58 abre a figura rotulada como (2), enquanto a linha 59 limpa a janela gráfica para iniciar uma nova visualização. A função `plot`, utilizada na linha 60, emprega as listas x_i (como eixo x) e tempo (como eixo y), conectando os pontos por linhas com marcadores em forma de círculos — especificados pelo argumento `'o-r'`. A linha 64 salva o gráfico como imagem, nomeando-o `figure2.png`, conforme o exemplo do código.

Quadro B5 - Implementação gráfica dos valores de x_i versus t, relativos à cada uma das trajetórias, e os cálculos usados para comparação com a Lei de Snell-Descartes.

Linha	Implementação
58	`figure (2);`
59	`clf;`
60	`plot(xi, tempo, 'o-r');`
61	`xlabel("xi (m)", "fontsize" ,12);`
62	`ylabel ("t (s)", "fontsize" ,12);`
63	`title("Tempo do percurso vs. Posição xi","fontsize",12)`
64	`saveas(2,"figure2 .png")`
65	`x0= ordem(1,1);`
66	`seno1 = abs((p(1) - x0) / sqrt((p(1) - x0)^2 + (p(2) - o(2))^2));`
67	`seno2 = abs((q(1) - x0) / sqrt((q(1) - x0)^2 + (q(2) - o(2))^2));`
68	`printf("x0 = %d\n", x0);`
69	`printf("seno1 / seno2 = %d\n", seno1/seno2 );`

Linha	Implementação
70	`printf("n2/n1 = %d\n", n2/n1);`

Na linha 65, é definida a variável `x0`, correspondente ao valor de x_i que resulta no menor tempo de percurso para a luz. Com base nos pontos P, Q e (x_0, 0), as linhas 66 e 67 calculam $sen\ (\theta_1)$ e $sen\ (\theta_2)$. Os catetos opostos são dados pelas distâncias $|P_x - x_0|$ e $|Q_x - x_0|$, enquanto as hipotenusas correspondem às expressões $\sqrt{(P_x - x_0)^2 + P_y^2}$ e $\sqrt{(Q_x - x_0)^2 + Q_y^2}$, respectivamente.

Por fim, os resultados são exibidos no console por meio da função `printf()` nas três últimas linhas: o valor de `x0` na linha 68; a razão $sen\ (\theta_1)/sen\ (\theta_2)$ na linha 69; e a razão entre os índices de refração n_2/n_1 na linha 70.

Os cálculos nas últimas linhas do script, além dos gráficos gerados, permitem comparar e verificar se o valor obtido para x_i está de acordo com a Lei de Snell.